Parametrized System Level Design: Real-Time X-Ray Image Processing Case Study

Tsvetan Shoshkov[1], Todor Stefanov[2], and Bart Kienhuis[2]
[1]Faculty of Electronics, Technical University of Sofia, Sofia, Bulgaria
[2]Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands
Email: tsh@tu-sofia.bg, t.p.stefanov@liacs.leidenuniv.nl, a.c.j.kienhuis@liacs.leidenuniv.nl

***Abstract*—Complex embedded systems are used to facilitate real-time data processing applications. To cope with their surroundings, these systems need to provide some form of parameterization. Developing such systems is challenging as the system needs to execute in a stable and consistent way with the parameterization. In this paper, we want to present how we can develop such a system from C-code using the Compaan technology and show a real-time medical X-Ray image processing case study. Our purpose is to evaluate the system level synthesis flow of Compaan and to assess if the Compaan technology is capable of realizing such complex parametrized system.**

I. Introduction

Complex embedded systems are used to facilitate real-time data processing applications. The embedded systems we are interested in typically have to process large amount of data that represents, for example, medical images, astronomy data, network traffic, or even stock exchange information. To cope with their surroundings, these systems need to provide some form of *dynamic parameterization*. For example, to adjust parameters of filters, to switch in or out particular computational blocks or to select regions for further analysis; all at runtime.

The development of such systems is challenging as a designer needs to develop a high-performing system that executes in a stable and consistent way with the parameterization. In this paper, we want to present how we can develop such a system using the Compaan technology and show a real-time medical X-Ray image processing case study. The Compaan technology realizes this design faster and more reliable than traditional design approaches that rely on more low level or adhoc development techniques to realize parameterization.

A. Problem Description

We assume that the general structure of a high-performance processing design is given in Fig. 1. It shows how an incoming data-stream (data_in) is processed by a number of blocks into an outgoing data-stream (data_out). These blocks perform functions on the data stream; an example would be a filter. The transformation from the input stream to the output stream is the data flow processing. This typically happens on huge data volumes in real time, i.e., the design can cope with the throughput of the incoming data stream as indicated by the "data flow" arrow. The blocks processing the stream can be configured by dynamic parameters. For example, the coefficients used in a Filter in Block 1 can be modified at runtime. In case of Block 2, it not only takes parameter values but also produces parameter values that could for example drive a gauge. Block N is presented dashed as the system may include a certain number of blocks. These blocks might be switched on and off at runtime depending on values measured by another block (e.g., Block 2) or under user control. The parameterization of the blocks is represented by the "control" arrow and describes the control flow. The control flow is operating at a much lower frequency than the data flow. It is considered orthogonal to the data flow as the control requires another view than the data flow.

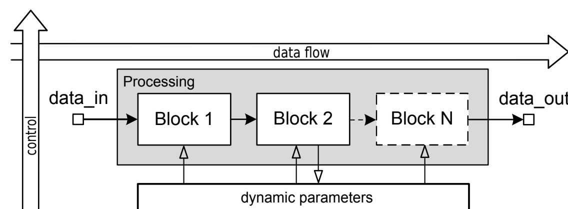

Fig. 1. General Processing Block Structure

So given a high-level description in C-code, the problem is to obtain a deep pipelined design that processes the incoming data with a certain throughput, while interacting in a consistent and reliable way with the control path.

B. Motivating Example

A real-time medical X-Ray image processing system uses the general processing block structure in Fig. 1 for its processing. Such a system is used for example when performing a vascular dottering procedure (balloon catheterization). In this procedure, a patient is laying on a table and the heart is exposed to a continuous X-Ray beam to produce a live image of the heart on a screen. A doctor inserts a catheter in a blood vessel and follows the way the catheter traverse to the heart on the screen until it reaches the location which requires dottering.

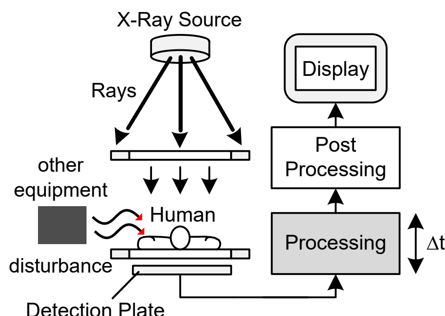

Fig. 2. Real-Time Medical X-Ray Image Processing System

The complete X-Ray set-up is shown in Fig. 2. It shows how X-Rays go from a X-Ray source through a human body

to a detection plate. The X-Ray hitting the detection plate form a raw image that is further processed by the processing phase and is projected on a display. A doctor performing the procedure watches this display.

In this paper, we are interested in the gray processing block in Fig. 2. This block is typically realized on a FPGA because real-time image processing is involved. A FPGA offers enough parallelism to handle the real-time requirements [1]. In realizing the processing block, a number of problems need to be solved. For example, unwanted artifacts must be filtered, system response time must be satisfied, and parameters must be used to make the system flexible.

There are unwanted artifacts in the output image due to other equipment present in the X-Ray engine. This equipment leads to magnetic disturbance that results into unwanted artifacts that show up as stripes on the image. Therefore, the image must be processed before it is transmitted further for post processing and displaying.

The response time of the processing system is very important and has to stay below a certain limit given by $\triangle_t$. In practice, this limit has to remain below 150 milliseconds. If a doctor moves a catheter, an update on the display needs to happen within this limit to make sure the image and the actual position of the catheter are the same.

The X-Ray machine is operating in different environments. For example, the contrast of an image of the X-Ray detection plate is sensitive to temperature. For correct processing, the medical personnel have to adjust the system to the operating temperature. Also, they need to be able to observe the image with or without certain filters applied. These modifications to the system, require the concept of dynamic parameters.

C. Paper Contributions

The main contributions of the work presented in this paper are:

- We present an existing technology (the Compaan technology) that is capable of producing a complete parameterized system solution on FPGAs from a C-code specification;
- We validate the solution on a simplified industrial X-Ray image processing system;
- We quantify that we meet requirements and discuss the FPGA solution in terms of robustness and consistency.

In the remainder of this paper, we present related work in Sec. II and background in Sec. III. In Sec. IV, we present a simplified version of a real industrial X-ray processing pipeline. We present experiments and analyze the results in Sec. V and concluded the paper in Sec. VI.

II. Related Work

The field of medical image processing benefits greatly from hardware acceleration. It allows that complex functions can be realized and operated in real-time in an embedded context. This makes new diagnostic machines possible to be used by a doctor. Besides the medical field, the use of FPGAs for acceleration is also relevant for the field of astronomy, finance, and networking. The overall problem is that the development process to program a FPGA is complex and takes too long. Also, the offered environment to program a FPGA requires hardware expertise which further limits the use of FPGAs.

To speed up development time, one can use High-Level Synthesis (HLS) tools [2]. HLS tools can convert C-code into efficient IP Cores as shown in [2]–[4]. HLS tools can implement hardware processing cores but do not provide full system integration. It is still the task of a designer to combine the IP cores into a complete system. The SDAccel tool of Xilinx [5] further automates the combination of IP cores into a system, but a designer still has to manually partition his design into a collection of tasks. This means a designer has to investigate the original sequential C-code and figure out opportunities for parallelization. HLS systems based on OpenCL [6] are able to handle in particular data level parallelism. Having its root in software development for GPUs, it is natural to generate a complete system. More advanced forms of task parallelism are still rudimentary in OpenCL let alone for hardware.

Using dataflow to express stream-based application is already done by many researchers for many years. Specialized languages exist to express the dataflow like StreamIt [7] and CAL [8]. StreamIt focuses on multi-core platforms while CAL targets FPGAs. In both cases, a designer needs to express its application directly into a dataflow format. The CASH tool [9] converts arbitrary C-code into a dataflow model that is subsequently expressed in hardware. The CASH tool, like the mentioned HLS tools, uses control dataflow graphs (CFDG) as the basis as opposed to process networks (PN) used by the Compaan technology.

In the case study of the medical X-Ray processing system, we want to use the Compaan technology. This technology can produce a complete system solution on FPGAs from a C-code specification. It performs the High Level Synthesis (HLS) required to realize the processing system on a FPGA. But, it also takes care of the synchronization between the data flow and control flow providing system consistency. The use of the process networks provides a framework to handle the parameterization of a design in a consistent way as compared to adhoc or low level solutions offered by other techniques.

III. Design Flow

The Compaan technology is developed at Leiden University and has resulted in a complete design flow to program FPGAs from C-code. The foundation work is presented in [10]–[13] and the first system concept was presented in [14]. The Compaan design flow is centered around the Compaan compiler [11] and its backend that is based on the ESPAM tool [12]. Fig. 3 presents a block diagram of the flow.

The design flow starts with an application written in sequential C-code. The application needs to be specified as a parameterized static affine nested loop program (SANLP). This is an important subset of the C language that does not allow for data dependent conditions or pointers. Although this sounds limited, many relevant applications can be specified by this subset.

The Compaan compiler automates the conversion of the input C code specification into a Process Network (PN) model of computation [15]. This is a parallel execution model in

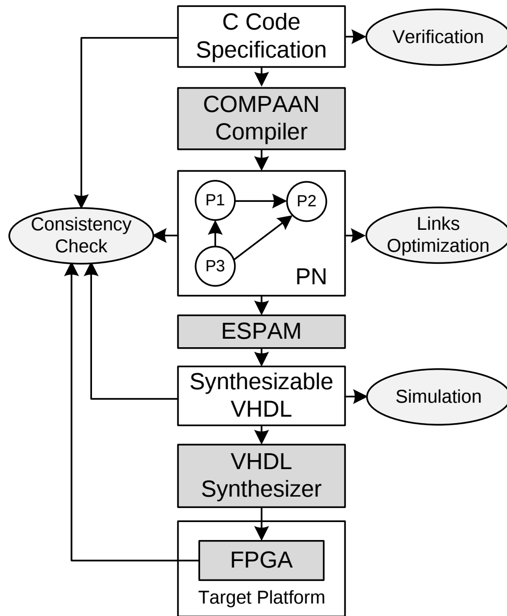

Fig. 3. Compaan Design Flow

which processes are connected by point-to-point communication links (i.e., FIFOs). A process may be a process on itself, or contain a hierarchical subnetwork. The communication links are given a particular buffer size such that the network will never deadlock. A process network is a data flow parallel representation of the original sequential C-code.

To obtain the parallel representation, the Compaan compiler employs polyhedral techniques to do the conversion from sequential C-code into the parallel PN representation. Compaan uses exact dataflow analysis to find all dependencies in the original C-code. Based on these dependencies, the original C-code is converted into processes and FIFO channels [11]. In the obtained PN, each process has its own notion of where it is located in the global execution. This location is derived from the for-loops in the original C-code and is exploited to determine precisely when it is safe to have control flow and dataflow interact with each other. Because of the localization, each process knows when a new frame or line starts and thus when it is safe to update parameter if needed.

The process network is converted into synthesizable VHDL code using ESPAM [12]. Each process is realized on HW using the LAURA processor model [13]. A LAURA processor consists of three separate blocks - read, execute and write. The read block waits until data tokens are ready to be read. The execute block processes the data. When tokens are ready to be sent, the write block writes them into the corresponding communication link if space is available. The execute block can integrate deep pipelined IP blocks that are hand designed or obtained from commercial High-Level Synthesis (HLS) synthesizes like Vivado-HLS or Catapult-C. Once all the VHDL is obtained for a process network, commercial tools realize a bit stream for one or more FPGAs.

The Compaan tool allows for simulation and verification of the system at each stage of development. Especially in the development of the tooling, this is a very important step. It allows us to check if all stages lead to equivalent input/output behavior. A process network can be simulated as a multi-threaded program. This way, Compaan validates that the parallel process network is input/output equivalent to the original sequential C-code. Furthermore, the obtained VHDL code can also be simulated to validate that it is again equivalent to the input/output relation of the original C-code.

IV. Case Study

In the case study, we implement a simplified real-time medical X-Ray image processing application based on an industrial design. The simplified processing block structure is given in Fig. 4, which is an instance of the general processing system given in Fig. 1. It shows that the incoming raw X-Ray image is first processed by the *Pixel Correction* block, the *EP filter* block, the *Gain* block, the *Zoom Window* block and finally the *Zoom Average* block.

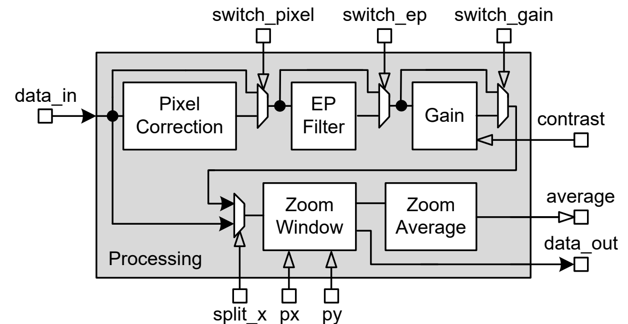

Fig. 4. X-Ray Image Processing Structure

A. System Components

The first block is the *Pixel Correction* block. The X-Ray detection plate in Fig. 2 typically has some broken pixels. This results in missing data in the X-Ray image. The *Pixel Correction* block restores the value of the missing pixel by taking the average value of four neighbor pixels.

The second element of the pipeline is the *EP Filter* block. Before X-Rays hit the detection plate, they received magnetic disturbance that results into stripes on the X-Ray image. The EP Filter is an application used in industrial X-Ray products to filter out these image artifacts. This filter is in itself a big and a complex piece of design.

The next element of the pipeline is the *Gain* block. The X-Ray image contrast is dependent on the operation temperature. Temperature variations are compensated by the Gain block that applies contrast adjustment to the image.

Doctors sometimes need to observe the image in larger scales to be able to make better judgment. The *Zoom Window* block provides such functionality. The doctor can move a window over the image with a mouse. The image inside the window is then enlarged two times.

The *Zoom Average* block is implemented to calculate the average value of the pixels inside the zoom window. This block is realized to show the capability of calculating certain important parameters of the image in real time and communicate this value to the outside world. We can attach for example a graph to the average parameter to see how the average values change from frame-to-frame for the X-Ray images.

B. System Parameters

Parameters are making the system flexible and capable of interaction with its environment. We defined 8 parameters in our X-Ray processing structure in Fig. 4. The parameters *switch_pixel*, *switch_ep* and *switch_gain* are used to allow a doctor to switch blocks on or off at run-time. We provide a *contrast* parameter to adjust the gain coefficient. Parameters *px* and *py* are used to move the Zoom Window to the location where more details must be visible. Parameter *split_x* is used to split the display image between the original raw input image and the processed image. This allows for a comparison between the two images to quickly assess the improvements in the output image. Parameter *average* is used to output the average value of all pixels in the zoomed window. This value is calculated for every frame.

C. Design Flow

To develop an actual implementation of the X-Ray processing structure in Fig. 4, we follow the steps of the Compaan Design Flow shown in Fig. 3.

1) Input Specification: The first step is to create the input SW specification. A small snippet of the C-code specification is presented in Listing 1. This snippet shows some concepts in Compaan like (de)-linearization, hierarchy, and parameterization.

Listing 1. Snippet of X-Ray processing block C-code specification

```
//Stream in data and initialize output
for (i = 0; i < frames * frame_size; i++) {
    img_1D[i] = data_in[i];
}
for (y = 0; y < img_height; y++) {
    for (x = 0; x < img_width; x++) {
        //Deserialize data 1D -> 2D
        img_2D[y][x] = img_1D[(img_width) * y + x];
    }
}
//Hierarchical call
if (switch_pixel <= 0) {
    img_pixel = ep_filter(img_2D);
} else {
    img_pixel = pass_filter(img_2D);
}
// split screen
for (y = 0; y < img_height; y++) {
    for (x = 0; x < img_width; x++) {
        if (x < split_x) {
            img_out[y][x] = img_2D[y][x];
            dump(img_pixel[y][x]);
        } else {
            img_out[y][x] = img_pixel[y][x];
        }
    }
}
```

The data comes in via variable `data_in` as a number of frames of a particular frame size. This 1-D array is deserialized into the 2-D array `img_2D`. Next, the complex `ep_filter` or the `pass_filter` function is applied, depending on the value of parameter `switch_pixel`. Then we iterate over all the pixels of the image. Based on the value of parameter `split_x`, we show either the original image (i.e., `img_2D`) or the processed data (i.e., `img_pixel`). If the original image is used, we need to dump the data of `img_pixel` to avoid buffers to fill up and cause deadlock. We use the function `dump` for this purpose. In the C-code, the `ep_filter` and `pass_filter` are expressed as hierarchical calls. This hides complexity and makes reuse of components possible. Especially `ep_filter` is a complex filter in itself expressed in C-code.

Compaan distinguishes between two types of parameter updates: synchronous and immediate. A *synchronous parameter* value is updated synchronously with, for example, the start of a new line or new frame. Such a start happens at different moments for the different nodes in the data flow graph, but Compaan uses the dataflow to make sure nodes update parameters at the appropriate moment. The parameters `split_x` and `switch_pixel` are examples of synchronous parameters. An *immediate parameter* update happens immediately; it synchronizes with nothing and no guarantee is given when the change is picked up. Of all parameters, the `contrast` parameter in Fig. 4 is an immediate parameter.

2) PN Model of Computation: The second step in the development of the X-Ray processing structure is to generate a PN with the Compaan compiler. For the complete sequential C-code (750 lines of C), the resulting PN is shown in Fig. 5. In this figure, edges represent communication channels, ellipses represent processes, and rectangular boxes represent hierarchical structures. For example, the rectangular box ND_12 is represented again by a process network as shown in Fig. 6.

TABLE I
NUMBER OF PROCESSES AND CHANNELS

PN	processes	channels
System	24	36
EP_Filter	10	118
Pixel_Filter	5	12
Gain_Filter	3	2
Pass_Filter	3	2

The complete hierarchical PN consists of 51 processes and 174 edges. In Table I, the number of processes and channels are given for the various PNs that make up the complete system. All edges have a size of 1, except for 8 edges. These edges have a size between 512 and 8196 pixels.

In a PN, all ellipses represent processes that run autonomously. All flow dependencies present in the original C-code specification are converted into FIFO communication links. This means that all memory accesses in the C-code are partitioned and serialized over 1-D channels. When looking at the C-code in Listing 1, it seems that the functions in the hierarchical call to the `ep_filter` or `pass_filter` can only execute when the complete array `img_2D` is present. However, since data is partitioned and serialized, the functions can already start as soon as the first data token appears; the array `img_2D` has, thus, become a stream.

3) Implementation: The third step is to select a target platform for our system and map the processes and channels onto the target platform resources. We have to select an interface to stream in and out the data on the platform. We use PCI Express (PCIe) for this purpose. Once the PN, the target platform, and the mapping specifications are ready, synthesizable VHDL code and the register file for the dynamic parameters is generated by ESPAM (see Fig. 3). To the synthesizable VHDL code, the PCIe interface is added and synthesized by any third party tools to a FPGA bit stream for our target platform.

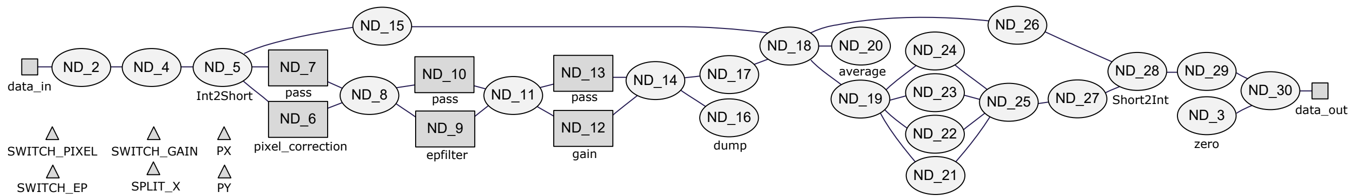

Fig. 5. System PN

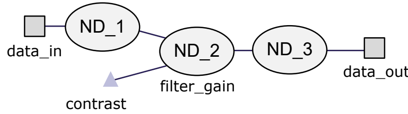

Fig. 6. ND_12 (Gain) as PN Sub-network

V. Evaluation

The main objective of our evaluation is to verify that we can make a parameterized design from C-code that satisfies the application requirements in an acceptable amount of development time. This is discussed in this section. We also highlight some particular features we found convenient in the process of developing the application.

A. Environment

The Compaan design flow provides full system integration. This means that it realizes the complete HW to program the FPGA and the hooks to integrate with PCIe and SW to set and read parameter values. To validate if the system works, we process raw x-ray images of a heart containing the visual artifacts (stripes). Each image consists of 806x806 samples of 16 bits. We used OpenCV to stream in and out the sequence of images. To set and read the parameters, we made a simple Java application. We connected GUI widgets to the parameters to provide user control. We used a check_box widget for the switch parameters, a slider for *x_split* and *contrast* and a mouse driver for the *px* and *py* parameters. We also added a Graph widget to show live the average value inside the zoom window. The resulting system is shown in Fig. 7. It shows the connections between a computer running the SW and the FPGA running the HW. The structure of the HW processing design is as shown in Fig. 4.

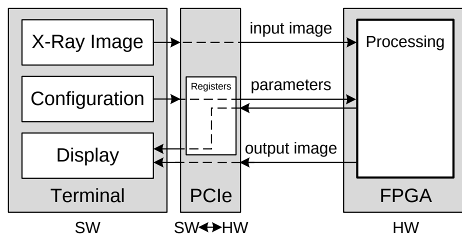

Fig. 7. System Environment

We use the Xilinx Virtex-6 FPGA ML605 evaluation board as our target platform. It is installed on a PCIe slot in a regular workstation. We use Xilinx ISE Design Suite 14.5 to simulate, synthesize, and program the target device. The OpenCV application is launched on the workstation to stream in and out the image data through the PCIe interface. Our PCIe driver uses DMA to move data to and from the ML605 evaluation board. The PCIe driver also accesses the registers for parameter values on the FPGA. These registers are set/-modified by the Java application at execution runtime.

B. Functional Requirements

To test our design as shown in Fig. 4, we use a sequence of 40 images with a moving phantom object that simulates a beating heart. It has horizontal stripes and bad pixels so we can functionality verify if our blocks can remove these artifacts. By observing the output image we verify that all stages of the design are working. We set parameters *switch_pixel*, *switch_ep*, and *switch_gain* to switch on and off the block *Pixel Correction*, *EP Filter*, and *Gain* respectively and observe if their impact is present or not in the output image. We also adjust the *contrast* parameter and we verify that it has the required impact on the image contrast. Moving a mouse, we are moving the position of the zooming window using the parameters *px* and *py* and we see the window moving in the output image. We set the parameter *split_x* and we see the comparison between the raw image and the processed one. Fig. 8 shows a frame of the output image. The image shows the difference between the input (left), the output data (right). It also shows the zoom window. The line in the middle is determined by the value for the *split_x* parameter in Listing 1.

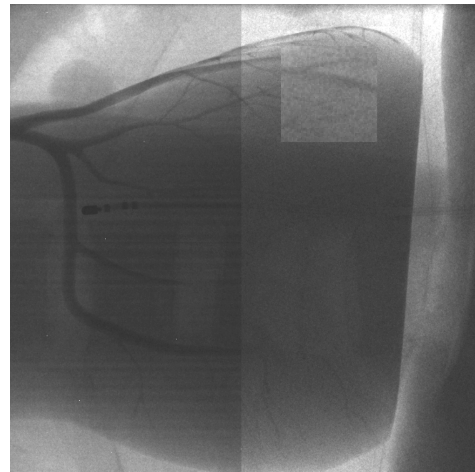

Fig. 8. Demo Output Image

C. Performance

The latency of the X-Ray processing structure in Fig. 4 is one of the main performance requirements. The latency between the input and the output of the processing structure must be in the range of tens of milliseconds. The complete system is synthesized at 200MHz and can process a new pixel every clock cycle. The observed latency of the design is 157.775 cycles, which translates into a $\triangle_t$ of 0.8 miliseconds which is way below the 150 miliseconds.

The system PN consists of 51 processes and hence 51 IP hardware blocks. One of these IP blocks has been realized with Vivado-HLS. This was the calculation of the average value of

the Zoom window. 7 IP blocks have been developed by hand; they are mainly used in the EP_Filter PN. The remaining 43 processes have an IP block that is so simple that ESPAM can generate the IP blocks. The final design without PCIe, takes 12.496 Slice LUT, 10.336 Slice Registers and 100 RAM components. The design can easily follow the throughput of the PCIe which is the bottleneck for the streaming speed.

The X-Ray processing structure in Fig. 4 has been developed in 2.5 man months. This includes the time necessary to simulate, implement and verify the design. We exclude the time consumed by modifications necessary to the Compaan compiler to generate the required design. Two persons have worked on the design; only one person had hardware skills. Most of the time is spent on modifying the input C-code.

D. Robustness

The designs made with the Compaan design flow are very robust. The use of automatic detection of parallelism and converting it into a VHDL model, leads to very robust designs. The use of the many parameters can lead to all kinds of synchronization issues. Nevertheless, they are all dealt with by the dataflow concepts used. We made a software application that modifies randomly all available parameters. This means that functions are switched on and off at random time and in a random order. Also the position of the mouse is constantly changed. We have run the design for two whole days and the system never crashed or deadlocked. Also, observe the switching of the `ep_filter` in Listing 1. The `ep_filter` has a pipeline latency of at least 806*18 clock cycles whereas the `pass_filter` has only 9 cycles. The dataflow handles switching between these large difference in pipeline depth without problems.

E. Localy Synchronous, Globally Asynchronous

Designs obtained by the Compaan design flow are always globally asynchronous and locally synchronous. Each ellipse in Fig. 5 works independently of others and communicate only via FIFO channels. The asynchronous behavior allows us to lock-step through a design and trace how data flows through the FIFOs. This improves much the observability of a design, making it easier to track how data moves through a design.

F. IP Integration

ESPAM only generates VHDL code for the processes and the connections between the processes. The synchronous IP block running inside a LAURA processor corresponding to a process has to be developed by hand or using a HLS synthesizer. Since each process is autonomous, each IP-block can be developed and verified independently before it is integrated into its LAURA processor and merged together with the other processors into the complete system design. This way it is easier to develop and verify its functionality avoiding any additional problems caused by the complexity of the whole design.

G. Memory Controller

In the conversion from sequential C-code into a process network, the global memory space is distributed over FIFO buffers (in this example 174). As a result, the implementation of the X-ray engine does not require a memory controller. All memory is distributed as FIFOs in the FPGA. If, however, large FIFOs are needed, they can be efficiently mapped on external memory as shown in [16].

VI. Conclusions

In this paper, we evaluated if the Compaan technology can generate a full system implementation in the medical domain from C-code. We have shown that we could implement the design that uses extensively parameterization to interact consistently with its environment. The synchronization between data flow and control is handled automatically and the final design is therefore very robust while satisfying the functional and timing requirements. The presented design automation will spare a lot of time and efforts, so developers can focus on developing the application itself in software and less on the complex hardware description.

References

[1] A. P. D. Binotto *et al.*, "A CPU, GPU, FPGA system for X-ray image processing using high-speed scientific cameras," in *25th International Symposium on Computer Architecture and High Performance Computing*, 2013, pp. 113–119.

[2] J. Cong *et al.*, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 2011.

[3] F. Winterstein, S. Bayliss, and G. Constantinides, "High level synthesis of dynamic data structures: A case study using vivado HLS," *International Conference on Field-Programmable Technology (FPT)*, 2013.

[4] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers, "Optimized generation of data-path from c codes for FPGAs," in *Design, Automation and Test in Europe (DATE)*, 2005, pp. 112–117.

[5] xilinx. (2014) The xilinx sdaccel development environment. [Online]. Available: http://www.xilinx.com/publications/prod_mktg/sdx/sdaccel-backgrounder.pdf

[6] altera. (2013) Implementing FPGA design with the OpenCL standard. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01173-opencl.pdf

[7] W. Thies, M. Karczmarek, and S. Amarasinghe, "Streamit: A language for streaming applications," in *In International Conference on Compiler Construction*, 2001, pp. 179–196.

[8] J. W. Janneck, I. D. Miller, D. B. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2009.

[9] G. Venkataramani *et al.*, "C to asynchronous dataflow circuits: An end-to-end toolflow," in *International Workshop on Logic synthesis (IWLS)*, Temecula, CA, June 2004, pp. 501–508.

[10] B. Kienhuis *et al.*, "Compaan: Deriving process networks from matlab for embedded signal processing architectures." in *8th International Workshop on Hardware/Software Codesign (CODES'2000)*, may 2000.

[11] A. Turjan, B. Kienhuis, and E. Deprettere, "Translating affine nested-loop programs to process networks," in *international conference on compilers, architecture, and synthesis for Embedded Systems (CASES'04)*, Washington D.C., USA, sept 2004.

[12] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and automated multi-processor system design, programming, and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 3, pp. 542–555, mar 2008.

[13] C. Zissulescu, T. Stefanov, B. Kienhuis, and E. Deprettere, "Laura: Leiden architecture research and exploration tool," in *International Conference on Field Programmable Logic and Applications (FPL)*, Lisbon, Portugal, sept 2003.

[14] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprettere, "System design using kahn process networks: The compaan/laura approach," in *Design, Automation and Test in Europe conference (DATE04)*, feb 2004.

[15] E. Lee and T. Parks, "Dataflow process networks," *Proceeding of the IEEE*, vol. 83, no. 5, pp. 773–801, may 1995.

[16] H. Nikolov, T. Stefanov, and E. Deprettere, "Efficient external memory interface for multi-processor platforms realized on FPGA chips," in *17th Int. Conference on Field Programmable Logic and Applications (FPL'07)*, 2007.